Better Forms with LiveView

Andrew Ek, Gig City Elixir 2023

Andrew Ek, Principal Engineer at Launch Scout

@ektastrophe on Twitter

andrew.ek@launchscout.com

https://github.com/andrewek/better-forms-with-liveview

LiveView gives us a *fantastic* developer experience

There's still a lot to learn about delivering excellent user experiences with LiveView

The most significant user interactions usually happen through *forms*

Our agenda for today:

1. Briefly characterize some good form practices

2. See how close we can get with LiveView *without* custom JS

3. Discuss when we might use (or not use) LiveView

Some reasonable form practices

Some reasonable form practices

Some reasonable form practices

1. Error messages should be close to the inupt

Some reasonable form practices

1. Error messages should be close to the inupt
2. The form should map to the user's mental model

Some reasonable form practices

1. Error messages should be close to the inupt
2. The form should map to the user's mental model
3. Forms should use labels, semantic HTML inputs

Some reasonable form practices

1. Error messages should be close to the inupt
2. The form should map to the user's mental model
3. Forms should use labels, semantic HTML inputs
4. Form should recover gracefully from error

Some reasonable form practices

1. Error messages should be close to the inupt
2. The form should map to the user's mental model
3. Forms should use labels, semantic HTML inputs
4. Form should recover gracefully from error
5. Errors should be shown at appropriate time

Some reasonable form practices

1. Error messages should be close to the inupt
2. The form should map to the user's mental model
3. Forms should use labels, semantic HTML inputs
4. Form should recover gracefully from error
5. Errors should be shown at appropriate time
6. We shouldn't disable the "submit" button if possible

Error Message Timing

We want to show errors as soon as we're sure it's an error, but not sooner.

Error Message Timing

Error Message Timing

1. Show all errors on submit

Error Message Timing

1. Show all errors on submit
2. Show per-field errors once the field has been touched and blurred

Error Message Timing

1. Show all errors on submit
2. Show per-field errors once the field has been touched and blurred
3. After submit or an error message for a field is shown, show all further error messages on change instead of on blur

How do we do this with LiveView?

Variation 1

Variation 1

```elixir
def handle_event("save", %{"invoice" => invoice_params}, socket) do
  changeset =
    invoice_params
    |> InvoiceContext.creation_changeset()

  result = InvoiceContext.insert(changeset)

  case result do
    {:ok, %Invoice{invoice_number: number}} ->
      socket
      |> put_flash(:success, "Created invoice ##{number}!")
      |> push_redirect(to: ~p"/invoices")
      |> then(&{:noreply, &1})

    {:error, changeset} ->
      socket
      |> put_flash(:error, "Something went wrong!")
      |> assign(changeset: changeset)
      |> then(&{:noreply, &1})
  end
end
```

Variation 1

```
<.form :let={f} for={@changeset} phx-submit="save">
  <div class="my-4">
    <%= label(f, :invoice_number) %>
    <%= text_input(f, :invoice_number) %>
  </div>

  <div class="my-4">
    <%= label(f, :recipient_email) %>
    <%= text_input(f, :recipient_email) %>
  </div>

  <% # .... %>

  <div class="my-4">
    <.button>Create Invoice</.button>
  </div>
</.form>
```

But!

We have some problems here...

Variation 1

Variation 1

1. Validation errors are far from their fields

Variation 1

1. Validation errors are far from their fields
2. Validation only on submit

Variation 1

1. Validation errors are far from their fields
2. Validation only on submit
3. Some inputs are hard to find valid value for (e.g. invoice number)

Variation 1

1. Validation errors are far from their fields
2. Validation only on submit
3. Some inputs are hard to find valid value for (e.g. invoice number)
4. Inputs require special knowledge

Variation 2

Variation 2

```
<div class="my-4">
  <%= label(f, :recipient_email) %>
  <%= email_input(f, :recipient_email) %>
  <%= error_tag(f, :recipient_email) %>
</div>
```

Variation 2

```
<div class="my-4">
  <%= label(f, :due_on) %>
  <%= date_input(f, :due_on) %>
  <%= error_tag(f, :due_on) %>
</div>
```

Variation 2

```
def mount(_params, _session, socket) do
  changeset =
    %{}
    |> Map.put(:invoice_number, InvoiceContext.recommended_invoice_number())
    |> InvoiceContext.creation_changeset()

  status_options = InvoiceContext.status_options()

  socket
  |> assign(page_title: "New Invoice - 2")
  |> assign(changeset: changeset)
  |> assign(status_options: status_options)
  |> then(&{:ok, &1})
end
```

Variation 2

```
<.button phx-disable-with="Saving...">
  Create Invoice
</.button>
```

Variation 2

Relative to version 1, we:

Variation 2

Relative to version 1, we:

1. Show errors in logical spot

Variation 2

Relative to version 1, we:

1. Show errors in logical spot
2. Provide a sensible default Invoice Number value

Variation 2

Relative to version 1, we:

1. Show errors in logical spot
2. Provide a sensible default Invoice Number value
3. Use `email_input` (`<input type="email">`) for email address

Variation 2

Relative to version 1, we:

1. Show errors in logical spot
2. Provide a sensible default Invoice Number value
3. Use `email_input` (`<input type="email">`) for email address
4. Handle slow-submit better

But!

We have some problems here...

Variation 2

Variation 2

1. Dev experience is clunky

Variation 2

1. Dev experience is clunky
2. Doesn't handle disconnects

Variation 2

1. Dev experience is clunky
2. Doesn't handle disconnects
3. Validations still only show on submit

Variation 3

Variation 3

```
def handle_event("validate", %{"invoice" => invoice_params}, socket) do
  changeset =
    invoice_params
    |> InvoiceContext.creation_changeset()
    |> Map.put(:action, :validate)

  socket
  |> assign(changeset: changeset)
  |> then(&{:noreply, &1})
end
```

Variation 3

```
<.form
  :let={f}
  id="invoice-3-form"
  for={@changeset}
  phx-submit="save"
  phx-change="validate"
>
  # ....
```

Variation 3

Relative to version 2, we:

Variation 3

Relative to version 2, we:

1. Show errors in real-time

Variation 3

Relative to version 2, we:

1. Show errors in real-time
2. Handle disconnect/reconnect much more gracefully

But!

We have some problems here.

Variation 3

Variation 3

1. Dev experience is still clunky

Variation 3

1. Dev experience is still clunky
2. Errors show on first keypress

Variation 3

1. Dev experience is still clunky
2. Errors show on first keypress
3. We don't yet help the user do the right thing

Variation 4

Variation 4

```
def mount(_params, _session, socket) do
  form =
    %{}
    |> Map.put(:invoice_number, InvoiceContext.recommended_invoice_number())
    |> InvoiceContext.creation_changeset()
    |> to_form()

  status_options = InvoiceContext.status_options()

  socket
  |> assign(page_title: "New Invoice - 4")
  |> assign(form: form)
  |> assign(status_options: status_options)
  |> then(&{:ok, &1})
end
```

Variation 4

```
def handle_event("validate", %{"invoice" => invoice_params}, socket) do
  form =
    invoice_params
    |> InvoiceContext.creation_changeset()
    |> Map.put(:action, :validate)
    |> to_form()

  socket
  |> assign(form: form)
  |> then(&{:noreply, &1})
end
```

Variation 4

```
<.input
  field={@form[:invoice_number]}
  label="Invoice Number"
  type="number"
/>
```

Variation 4

```
<.simple_form
  for={@form}
  id="invoice-form"
  phx-change="validate"
  phx-submit="save"
>
  < inputs >

  <:actions>
    <.button phx-disable-with="Saving..." type="submit">Create</.button>
  </:actions>
</.simple_form>
```

Variation 4

```
def input(assigns) do
  ~H"""
  <div phx-feedback-for={@name}>
    <.label for={@id}><%= @label %></.label>
    <input
      type={@type}
      name={@name}
      id={@id}
      value={Phoenix.HTML.Form.normalize_value(@type, @value)}
      class={your_list_of_classes}
      {@rest}
    />
    <.error :for={msg <- @errors}><%= msg %></.error>
  </div>
  """
end
```

Variation 4

Relative to version 3, we:

Variation 4

Relative to version 3, we:

1. Show errors in a more timely fashion

Variation 4

Relative to version 3, we:

1. Show errors in a more timely fashion
2. Have much improved developer ergonomics with `to_form()` and input components

But!

We have some problems here.

Variation 4

Variation 4

1. Dirty/clean state isn't quite right

Variation 4

1. Dirty/clean state isn't quite right
2. We don't yet help the user do the right thing

Variation 4

1. Dirty/clean state isn't quite right
2. We don't yet help the user do the right thing
3. Accessibility isn't great

Variation 4

1. Dirty/clean state isn't quite right
2. We don't yet help the user do the right thing
3. Accessibility isn't great
4. Still run uniqueness query on every change

Variation 5

Variation 5

```
<.input
  field={@form[:invoice_number]}
  label="Invoice Number"
  type="number"
  required
  min={1}
/>
```

Variation 5

```
<.input
  field={@form[:recipient_email]}
  label="Recipient Email"
  type="email"
  required
/>
```

Variation 5

```
<.input
  field={@form[:status]}
  label="Status"
  type="select"
  options={@status_options}
  required
/>
```

Variation 5

Relative to version 4 we:

Variation 5

Relative to version 4 we:

1. Use attributes on the HTML inputs to help user

Variation 5

Relative to version 4 we:

1. Use attributes on the HTML inputs to help user
2. Have better accessibility

But!

We have some problems here.

Variation 5

Variation 5

1. `amount_in_cents` doesn't track with user mental model

Variation 5

1. `amount_in_cents` doesn't track with user mental model
2. Still haven't tackled line items

Variation 5

1. `amount_in_cents` doesn't track with user mental model
2. Still haven't tackled line items
3. Email validation is *aggressive*

Variation 6

Variation 6

```
<.input
  field={@form[:recipient_email]}
  label="Recipient Email"
  type="email"
  required
  autofocus
  phx-debounce="blur"
/>
```

Variation 6

```
<.input
  field={@form[:amount_in_dollars]}
  label="Amount ($)"
  type="number"
  required
  step={0.01}
  min={0.01}
  placeholder="0.00"
/>
```

Variation 6

```
def create_with_dollars_changeset(attrs) do
  %Invoice{}
  |> validate_amount_in_dollars(attrs)
  |> validate_description(attrs)
  |> validate_due_on(attrs)
  |> validate_invoice_number(attrs)
  |> validate_recipient_email(attrs)
  |> validate_status(attrs)
end
```

Variation 6

```
defp validate_amount_in_dollars(changeset, attrs) do
  changeset
  |> cast(attrs, [:amount_in_dollars])
  |> validate_required([:amount_in_dollars])
  |> validate_number(:amount_in_dollars, greater_than: 0)
  |> cast(%{
    amount_in_cents: to_cents(attrs[:amount_in_dollars])},
    [:amount_in_cents]
  )
end
```

Variation 6

Relative to version 5 we:

Variation 6

Relative to version 5 we:

1. Handle email validation in a more friendly way

Variation 6

Relative to version 5 we:

1. Handle email validation in a more friendly way
2. Use `amount_in_dollars` virtual attribute to match user model

But!

We have some problems here.

Variation 6

Variation 6

1. We still haven't tackled line items

Variation 6

1. We still haven't tackled line items
2. There are still possible latency issues

Variation 6

1. We still haven't tackled line items
2. There are still possible latency issues
3. Feedback is still kind of weird sometimes

Variation 6

1. We still haven't tackled line items
2. There are still possible latency issues
3. Feedback is still kind of weird sometimes
4. Invoice number validation runs on every change (we'll come back to this)

Variation 6

```elixir
def handle_event("validate", %{"invoice" => invoice_params}, socket) do
  # Simulate some latency
  :timer.sleep(400)

  # ... the rest of the stuff
end
```

Closing Thoughts

Forms are *still* the main way users work with system we build.

Validation messages and feedback on submit are the main way users get feedback from forms

We can choose the level of polish we want to give to the user experience

Predictable is easier to get than perfect and a better investment for most teams

There are still some limitations to what LiveView gives us in the form user experience

There are times where it makes sense to use LiveView for forms, and times it doesn't (yet)

It probably makes sense if:

It probably makes sense if:

1. Your form will almost always have valid input

It probably makes sense if:

1. Your form will almost always have valid input
2. You can defer expensive validations until later

It probably makes sense if:

1. Your form will almost always have valid input
2. You can defer expensive validations until later
3. Network connectivity isn't a concern

It probably makes sense if:

1. Your form will almost always have valid input
2. You can defer expensive validations until later
3. Network connectivity isn't a concern
4. The form isn't overy complex or can be broken into simpler pieces

It might *not* make sense if:

It might *not* make sense if:

1. Your form is particularly dynamic or particularly complex

It might *not* make sense if:

1. Your form is particularly dynamic or particularly complex
2. Network connectivity is an issue

It might *not* make sense if:

1. Your form is particularly dynamic or particularly complex
2. Network connectivity is an issue
3. Your form cannot be made simpler

It might *not* make sense if:

1. Your form is particularly dynamic or particularly complex
2. Network connectivity is an issue
3. Your form cannot be made simpler
4. You have very special validation rules

Managing Very Complex Forms

Managing Very Complex Forms

Split changesets so that your initial validation is inexpensive but validation at insert is heavy-duty

Managing Very Complex Forms

Break very complex forms into smaller forms and normalize data -- instead creating one record, you have a parent record with many child records, each with their own form.

Managing Very Complex Forms

Bring in strategic Javascript (*Real Time Phoenix* describes how those JS elements might communicate with the server), possibly through `live_elements`

fin

Andrew Ek, Principal Engineer at Launch Scout

@ektastrophe on Twitter

andrew.ek@launchscout.com

https://github.com/andrewek/better-forms-with-liveview